Premier 3

The Premier Reference Manual

Circle Software

Premier 3

The Premier Reference Manual

Circle Software

Premier 3.1

The Premier Reference Manual.

First edition, March 1990
Second edition, July 1990

All rights reserved. No part of this manual may be reproduced in any form whatever, without the written consent of *Circle Software*. The only exceptions are as provided by the Copyright (Photocopying) Act, for the purpose of review, or for any software listed herein to be entered into a computer for the sole use of the owner of this book.

While every care has been take in the design and content of this book, and the software to which it refers, no responsibility can be accepted by *Circle Software* for any loss of data, time or money, or other consequential effects resulting from it's use, however caused.

While the command language described remains the property of Circle Software, use of the language by third parties to create Premier applications for publication is encouraged. The sale of such applications is subject to quality approval by Circle Software.

Premier is the copyright of -

Circle Software,
33, Restrop View,
Purton, Swindon,
Wilts. SN5 9DG
Tell: 0793 770021

Contents

Section 1 Names and Constants 1

Names. 1
Integer Constants 1
String Constants. 2

Section 2 Operators 5

Mathematical Operators. 5
Assignment Operators. 5
Logical Operators 6
String Operators. 6
Conversion Operators 7

Section 3 System Constants 9

Section 4 Variables & Functions 11

Section 5 Command Words 83

Section 6 Key Codes 197

Index 203

Section 1 Names & Constants

This section describes the syntax of user definable names, and the permitted form of string and integer constants.

The syntax descriptions below use the following nomenclature.

<item>	A single instance of the named item.
[...]	A list of optional items.
| (bar)	Meaning 'or' within an optional list.
*n	From zero to n repetitions of an item.

Names.

A Premier item name is formed as follows.

length	Up to 32 characters.
syntax	<letter> [<letter> | <digit> | $ | . | _] *31
	where <letter> may be upper or lower case.
	and where the case is significant.

In other words it should start with a letter, and may be followed by any sequence of letters, digits, the $ sign, underscore or the full stop, and is case sensitive.

Integer Constants

These may be entered in either decimal or hexadecimal notation, but both types must start with a digit. The formal definitions are:-

	dec = <digit> <digit>*9	i.e. 10 digits max.
	hex = 0x <hex> <hex>*7	i.e. 8 digits max.

where	<digit>	=	[0-9],
&	<hex>	=	[0-9 | a-f | A-F]

i.e. upper or lower case hex digits are allowed.

Some valid examples of each include:-

```
3,  1234567890, 007          Decimal
0x27,  0xFFF,   0x7f         Hex.
```

N.B. The 0 in 0x is a *zero*, not the letter 'O'

String Constants.

String constants must be enclosed by double quotes, as in:-

```
"This is a string constant."
```

Such a string may have a length from zero to 255 characters, and may include any ASCII value from zero to 255.

A formal definition is:-

```
string    =  " [ cha*255] "
cha       =  ( char | space | esc_cha )
char      =  all visible characters except '\' & ""
esc_seq   =  \ [ esc_cha | num_cha ]
esc_cha   =  a | b | f | n | r | t | v | \ | " | @
num_cha   =  dec_cha | hex_cha
dec_cha   =  digit [ digit*2 ]
hex_cha   =  x hex [ hex ]
digit     =  ( 0-9 )
hex       =  ( 0-9 | a-f | A-F )
```

In English, this means that a string may normally consist of any printable character, except the backslash and the double quote. Other characters may be included by using an *escape sequence*. This is identified by the backslash, followed by either a special escape character, or a number. The number may be up to three decimal digits, or an 'x' followed by up to

two hex digits.

The escape characters may be one of the following, shown here including the backslash for clarity.

Char.	ASCII Value.
\@	zero, null.
\a	7, bell
\b	8, backspace
\t	9, tab
\n	10, new line
\f	12, form feed
\r	13, return
\\	92, a backslash
\"	34, a double quote

The last two can only be included as escape characters, since they otherwise have special meanings, and cannot be included literally. The numeric escape sequences are as follows.

\n	The ASCII value n, decimal.
\xn	The ASCII value n, hex.

A maximum of 3 decimal and two hex digits are allowed, the maximum value being 255, or FF.

Here are some examples:-

```
"This ends with a newline.\n"
"This contains \"quotes\""
"This includes 'esc' 'R' - \27R"
"\xff"
```

N.B. Since an embedded hex number may contain up to two digits, watch out for sequences such as:-

```
"\x7Friday"
```

where the 'F' will wrongly be translated as part of the hex number 7F. Use `"\x07Friday"`.

In the above manner, a string may contain any value from zero to 255. The non printing characters are mainly required for specifying printer control codes, etc.

Section 2 Operators

This section describes the range of operators available within Premier. For details on how they are used, you should refer to the User Guide.

In each case the priority value is shown. This indicates the order in which the operators are applied. When several are used within an expression, the higher numbered operations are carried out first.

Mathematical Operators.

+	Addition	12
-	Subtraction	12
*	Multiplication	13
/	Division	13
%	Remainder	13
-	Negation (unary)	14
( )	Sub-expression delimitors. The contents of these are always given higher priority.	

Assignment Operators.

=	String or Integer	2
+=	String or Integer	2
-=	Integer	2
*=	Integer	2
/=	Integer	2

The compound operators, such as `x += a`, are equivalent to `x = x + a`

Logical Operators

Logical operators may be applied to integer or string values, and result in a value of *true* or *false*.

NOT	Negation, (unary)	14
==	Equal to	9
<>	Not equal to	9
>	Greater than	10
<	Less than	10
>=	Greater than or equal to	10
<=	Less than or equal to	10
AND		5
OR		4

N.B. The operators AND, OR & NOT are case insensitive.

String Operators.

+	Concatenation	12
+=	Extension	12
=	Centre justify	14
<	Left justify	14
>	Right justify	14

These are used as <integer> <operator> <string> to justify the string in a field width given by the integer, which may be an integer expression within parentheses.

e.g. `20>"text"`
or `25<date` where *date* is the system variable.

Conversion Operators.

These are unary operators, and convert the following item as described.

$	Integer to string	14
%	Integer/100 to string.	14
#	String to integer.	14
| (bar)	Int. constant to string.	14
&	Int. to hex string.	14
£	Int. to money.	14
?	Int. to Roman, lower case	14
?+	Int. to Roman, upper case	14

e.g.

```
$(5*x)
|34
£(a*vatrate)
"Page " + ?page
```

Section 3 System Constants

This chapter lists the various constants defined within Premier, which may be referred to by the user, but may not be altered. These constants are provided for convenience, for use in situations where an integer value is required by a command to indicate some predefined choice.

For instance, the colour command requires specific values to represent specific colours. Using these constants, the user need not remember the values required for each colour, but use a constant such as *red*, *blue*, etc. instead.

Some values are stated in hexadecimal where appropriate, as indicated by the '&' prefix.

Some commands require a *true* or *false* value, and these may be provided using the appropriate constant. Within Premier, a value of *zero* is taken to mean *false*, while *any non-zero* value is taken to mean *true*. The actual value assigned to the constant *true*, is not given, and should not be assumed.

The values *on* and *off*, are synonymous with *true* and *false*, and may be used where the meaning is clearer.

Constant	Value	Constant	Value
and	*false*	premdata	&D24
black	7	premdoc	&D26
blue	15	red	11
bold	&10000	right	0
command	&FFE	serious	4
down	2	subscript	&80000
equal	3	superscript	&100000
error	3	text	&FFF
false	*zero*	true	*non-zero*
fatal	5	underline	&20000
greaterthan	2	up	3
green	10	verbose	0
grey	4	warning	2
info	1	yellow	9
insert	*false*		
inverse	&20000		
italic	&40000		
left	1		
lessthan	1		
newline (*string*)	ASCII 10		
newpage (*string*)	ASCII 12		
notequal	4		
obey	&FEB		
off	*false*		
on	*true*		
or	*true*		
orange	14		
over	*true*		
premcmd	&D25		

Section 4 Variables & Functions

This section presents a full list of the pre-defined Premier variables and functions, one per page, in alphabetical order for quick reference.

Each page shows the item name, followed by it's type, i.e. integer or string, variable or function. Where a function is given, the item name includes the syntax for any parameters required. The following nomenclature is used.

<name>	Angle brackets indicate items that the user is expected to supply, and will usually be either an integer or string expression as explained. The brackets are NOT part of the item, and should NOT be supplied.
[...]	Square brackets are used to indicate optional items. Again, the brackets are not part of the item.

Where function parameters are optional, the parenthesis should also be omitted.

The examples given are provided only as illustrations of how the items may be used, and are not necessarily fully working statements. For instance, they may assume the existence of other variables or macros.

Item: buffer

Type: String variable

Value: The name of the buffer to which the currently selected marker belongs.

Example:
```
save buffer "SaveFile"
```

Explanation: The current buffer is saved in the given file.

Notes: If no marker is selected, this variable is assigned the null string ""

See Also: marker, window, filename

Item: character

Type: String variable.

Value: The character pointed to by the currently selected marker.

Example:
```
while ( character <> " " )
        cursor right
```

Explanation: The currently selected marker is moved right until a space is found.

Notes:

See Also: deletedchar

Item: characters[(<buffer>)]

Type: Integer function

Value: The number of characters in the named buffer.

Example:

```
show characters("PasteBuffer")
```

Explanation: The number of characters currently in the *PasteBuffer* is displayed.

Notes: The parameter (including parenthesis) is optional, and if not supplied, the number of characters in the currently selected buffer is given.

See Also:

Item: colour

Type: Integer variable

Value: The colour number of the current font.

Example:
```
if (colour<>black) colour black
```

Explanation: If the current font is not black, it is set to be so. This example also uses the command *colour*, and would be part of a dynamic item.

Notes: The value of *colour* is the colour of the font at the position of the dynamic item containing it, the colour of the font at the position of the currently selected marker, or -1. (zero to white)

See Also: font, width.

Item: column

Type: Integer variable

Value: The column number of the currently selected marker.

Example:

```
column = column + 5
```

Explanation: The current marker is moved 5 character positions to the right, if possible.

Notes: The above example uses the *column* command, it does not re-assign the system integer column.

If given from within a dynamic item, gives the column position where the item is inserted.

See Also: line, page, pageline.

Item: content(<directory>)

Type: String function

Value: A string containing all files found within the given directory, separated by commas.

Example:

```
menu "Files" = content(DocDir)
```

Explanation: A menu is created, offering a list of file names.

Notes: *DocDir* is assumed to contain a full directory path name.

The menu created could be used with the *choose* command, to allow the user to choose a file.

See Also:

Item: datafile

Type: String variable

Value: The name of the currently selected data base file, if any.

Example:

```
str lastfile = datafile
```

Explanation: A new string variable lastfile is assigned the name of the current data file. This would allow a return to the original file, after some operation on a new one.

Notes: If no data file is currently selected, this variable is assigned the null value ""

See Also: filename.

Item: date

Type: String variable

Value: A string representing the current date.

Example:
```
insert date
```

Explanation: The current date is inserted at the current marker position.

Notes: The date format may be changed by re-defining the system variable *dataformat*.

The default format is, e.g. 8th March 1990

See Also: time,dateformat,week,weekday,month,year.

Item: dateformat

Type: String variable

Value: A string defining the exact format of the date string assigned to the system string date.

Example:
```
dateformat = "%DY-%MN-%YR"
```

Explanation: The date format is set to produce a date of the form "16-06-90"

Notes: Full details of the special formatting characters used with this variable are given in a separate chapter, and in the Premier User's Guide.

See Also: date, time.

Item: defined(<object>)

Type: Integer function

Value: *True* or *false*, indicating whether the named object exists or not.

Example:

```
if ( defined("s") )
   s += ",more"
else
   str "s" = "some"
```

Explanation:

Notes: If the string variable s exists, ",more" is appended to it, otherwise a new string is created.

This function will return true whatever type the object is, as long as it exists.

See Also:

Item: deletedchar

Type: String variable

Value: The character last deleted using the delete command.

Example:

```
key 461 = insert deletedchar
```

Explanation: The Insert key is defined to re-insert the last character deleted.

Notes: If no character has yet been deleted, a null string is returned.

See Also: character

Item: direction

Type: String Variable.

Value: Either "forward" or "backward", representing the current find and search direction.

Example:

```
str "lastway" = direction
```

Explanation: A new string variable *lastway* is assigned the current search direction. This enables the previous direction to be restored, for instance when a search is completed in the other direction.

Notes:

See Also:

Item: end

Type: Integer Variable.

Value: Either *true* or *false*, indicating whether the current marker is at the end of it's buffer.

Example:

```
repeat CheckChar until end
```

Explanation: The macro *CheckChar* is repeated until the end of the buffer is reached.

Notes: A value of zero represents *false*, while any non-zero represents *true*.

See Also:

Item: errors

Type: Integer Variable

Value: The number of messages of error, or higher status, issued.

Example:
```
repeat MyMacro until errors
```

Explanation: The macro *MyMacro* is repeated until an error occurs.

Notes: The error counts are initialised before a command is executed

The value of errors should be taken as *true* or *false*.

N.B. Do not confuse this with error, (singular).

See Also: warnings.

Item: extract(<string>,<index>,<seps>)

Type: String function

Value: Returns the *index* numbered sub-string from *string*, where sub-strings are separated by any character from *seps*..

Example:
```
s = extract(filename,0,".")
```

Explanation: String variable *s* is assigned the *leaf name* from *filename*. That is, the *last* sub-string separated by dots.

Notes: If *index* is zero, the last sub-string is returned. An invalid *index* returns a null string, without error.

i.e. if `s = "this,that,other"`
then `extract(s,2,",")`
returns `"that"`

See Also: substring

Item: fieldcount

Type: Integer Variable.

Value: The number of fields in one record of the current data base.

Example:
```
while ( n < fieldcount ) {...}
```

Explanation: The loop is executed as long as `n` is less than the number of fields.

Notes: If there is no data base file specified, this variable will have a value of zero.

See Also: fieldvalue, fieldname, fieldlength.

Item: fieldlength(<field>)

Type: Integer Function.

Value: The length of the data base field number *field.*

Example:

```
length = fieldlength(5)
```

Explanation:

The integer variable length is assigned the length of field 5, in the current data base file.

Notes:

If no data base file is open, or the specified field does not exist, the function will have a value of zero.

The value <field> may be any integer expression.

See Also: fieldcount, fieldname, fieldvalue.

Item: fieldname(<field>)

Type: Integer Function.

Value: The name of the field number *field*, in the current data base file.

Example:
```
prompt fieldname(n), data
```

Explanation: The user is prompted with the name of a data base field, and the reply assigned to the variable *data*.

Notes: The integer *field* may be any integer expression evaluating to a value indicating the required field number.

The field names are those read from the first data base record, and used to form the automatic field string variables.

See Also: fieldvalue, fieldcount, fieldlength.

Item: fieldvalue(<field>)

Type: String Function.

Value: The contents of field number *field* from the current data base record.

Example:

```
fieldvalue(6) = "New value"
```

Explanation: A new value is assigned to field 6 of the current record.

This is not written to the disc until the command *write* is given.

Notes: Field values may be referenced by this function, or by the automatic string variables created for each data base file, having names taken from the first record.

See Also: fieldname, fieldlength, fieldcount.

Item: filename

Type: String Variable.

Value: The file name associated with the current buffer.

Example:
```
save buffer filename
```

Explanation: The current buffer is saved as it's assigned file name.

Notes: The file name associated with a buffer will be a full path name if the file was loaded either by name, or by dragging an icon from a directory viewer. Otherwise, and for new buffers, the file name will be the same as the buffer name.

See Also: filetype

Item: filetype

Type: Integer Variable

Value: The file type number of the current buffer.

Example:
```
if ( filetype == premdoc ) ...
```

Explanation: The following command is executed if the current buffer is a Premier document type.

Notes: The file types are as assigned by Acorn.

See Also: premdoc, promcmd, premdata, text, obey.

Item: font

Type: Integer variable

Value: A value containing the bit settings for the current font.

Example:
```
temp = font
```

Explanation: May be used within a dynamic item, to save the current font, while changing fonts temporarily, perhaps within a header of footer.

Notes: The value of *font* refers to the font at the position of the dynamic item containing it, the position of the currently selected marker, of zero.

See Also: colour, width.

Item: fontbold

Type: Integer Variable.

Value: Either *true* or *false*, indicating whether the current screen font is of type bold.

Example:

```
if ( fontbold ) send PFB
```

Explanation: If the current screen font is bold, the string *PFB* is sent to the printer.

Notes: The value of this variable is only meaningful when used within a dynamic text item.

See Also: fontitalic, fontunderline, etc.

Item: fontitalic

Type: Integer Variable

Value: Either *true* or *false*, indicating whether the current screen font is of type italic.

Example:
```
if ( fontitalic ) send PFI
```

Explanation: If the current screen font is italic, the string *PFI* is sent to the printer.

Notes: The value of this variable is only meaningful when used within a dynamic text item.

See Also: fontbold, fontunderline, etc.

Item: fontsubscript

Type: Integer Variable

Value: Either *true* or *false*, indicating whether the current screen font is of type subscript.

Example:
```
if ( fontsubscript ) send PFSB
```

Explanation: If the current screen font is italic, the string *PFSB* is sent to the printer.

Notes: The value of this variable is only meaningful when used within a dynamic text item.

See Also: fontbold, fontunderline, etc.

Item: fontsuperscript

Type: Integer Variable

Value: Either *true* or *false*, indicating whether the current screen font is of type superscript.

Example:
```
if (fontsuperscript) send PFSP
```

Explanation: If the current screen font is italic, the string *PFSP* is sent to the printer.

Notes: The value of this variable is only meaningful when used within a dynamic text item.

See Also: fontbold, fontunderline, etc.

Item: fontunderline

Type: Integer Variable

Value: Either *true* or *false*, indicating whether the current screen font is of type underline.

Example:
```
if ( fontunderline ) send PFU
```

Explanation: If the current screen font is italic, the string *PFU* is sent to the printer.

Notes: The value of this variable is only meaningful when used within a dynamic text item.

See Also: fontbold, fontsubscript, etc.

Item: footer

Type: String variable

Value: The current name used for page footers.

Example:

```
if ( footer == "FOOTER" )
   footer = "MyFoot"
```

Explanation:

If the current footer name is equal to the default, then it is re-set.

Notes: This function will return either "FOOTER", or a name specified using the *footer* command.

See Also: header.

Item: found

Type: Integer Variable.

Value: *True* or *false* indicating the result of the latest find, search, seek, or read operation.

Example:
```
if ( found ) replace newstring
```

Explanation: If the latest *find* or *search* operation was successful, then the text found is replaced with the value of *newstring*.

Notes: This variable may also be used to detect the end of a data base record, or the result of a *seek* operation on a data base file.

See Also: foundstring, end.

Item: foundstring

Type: String Variable.

Value: The result of the last find or search operation.

Example:
```
repeat Look until foundstring==target
```

Explanation: The macro *Look* is repeated until the string target is found. The macro is assumed to execute a find or search command.

Notes: This is most useful in conjunction with the *search* command, when wild cards are used, and the exact string found is not known in advance.

See Also: found.

Item: header

Type: String variable

Value: The current name used for page headers.

Example:

```
if (header=="HEADER") header="MyHead"
```

Explanation: If the current header name is equal to the default, then it is re-set.

Notes: This function will return either "HEADER", or a name specified using the *header* command.

See Also: footer.

Item: index(<string>,<sub-string>,<seps>)

Type: Integer function

Value: The index, or position, of *sub-string* within *string*, where sub-strings are separated by any character from *seps* (separators).

Example:
```
i = index(List, "Item", ",;")
```

Explanation: The integer variable *i* is assigned the position of the string "Item" within the definition of string variable *List*. Items are assumed to be separated by commas or semi-colons.

Notes: The first sub-string is numbered 1. If no such sub-string can be found, *zero* is returned.

See Also: substring, extract, locate.

Item: length(<string>)

Type: Integer function

Value: The length of the given string.

Example:

```
if ( length(s) > rightmargin-
    column ) insert newline
```

Explanation:

If there is insufficient room for string *s* on the current line, a *newline* is inserted.

Notes:

See Also:

Item: leftmargin

Type: Integer Variable

Value: The column number of the left margin, as set for the current buffer.

Example:
```
margins = leftmargin + 5, 65
```

Explanation: The left margin is increased by 5, and the right margin set to 65.

Notes: This variable *always* supplies the left margin as set for the whole buffer, not the margin at any specific point in the text.

See Also: rightmargin, page

Item: lines[<buffer>]

Type: Integer function

Value: The number of lines in the given buffer.

Example: show lines("Journal")

Explanation: The number of lines in the Journal buffer is displayed.

Notes: If the parameter is not given, the number of lines in the currently selected buffer, if any, is given.

See Also: words, characters.

Item: line

Type: Integer Variable

Value: The line number of the current marker.

Example:

```
line = line + windowheight
```

Explanation: The current marker is moved down by the height of the current window.

Notes: This example uses the *line* command, it does not re-assign a value to the line variable.

Lines are numbered from zero, and continue through header, footer, and perforation skip areas.

See Also: column, windowheight.

Item: locate(<string>,<sub-string>,<position>)

Type: Integer function

Value: The position of *sub-string* within *string*, searching from *position*.

Example:

```
i = locate(s,"*",5)
```

Explanation: Integer *i* is assigned the position of the first asterisk found after the fifth character.

Notes: This function always returns the character position from the *start* of the string, and searches from the character *after* the position given.

To search from the beginning, position should be *zero*.

Repeated searches may be made, by making *position* equal to the last returned value.

See Also: substring.

Item: marked

Type: Integer variable

Value: *True* or *false*, indicating if a marked text area has been defined.

Example:

```
if (marked) cut marker
```

Explanation: If a marked area exists, it is cut out and inserted at the current marker position. (See the *cut* command)

Notes: This variable should always be tested, prior to manipulating marked areas.

See Also:

Item: marker

Type: String Variable.

Value: The name of the currently selected marker.

Example:
```
if ( marker <> buffer ) goto marker
```

Explanation: If the current marker is not the default marker for the buffer, the *goto* command is used to make it so.

Notes: The default marker has the same name as it's buffer.

See Also: buffer, window, marked.

Item: messages

Type: Integer Variable.

Value: The current message reporting level.

Example:

```
int "level" = messages
messages = verbose
```

Explanation:

The current message reporting level is saved, before selecting to receive verbose messages. This allows the original level to be re-selected later.

Notes: This technique is useful when debugging a command file.

See Also: verbose, error, etc.

Item: month

Type: String variable

Value: The name of the current month.

Example:
```
show month
```

Explanation: the current month is displayed.

Notes: The month is always the full name, i.e. "September"

See Also: week, weekday, year.

Item: mousebuffer

Type: String Variable.

Value: The name of the buffer assigned to the window over which a mouse button was last clicked.

Example:
```
key 256 = select mousebuffer
```

Explanation: The mouse *select* button is defined to select the default marker belonging to the buffer displayed in the window clicked over.

Notes: The default marker is the marker created when a buffer is created, which has the same name as the buffer.

See Also: mousewindow, mouseline, mousecolumn.

Item: mousecolumn

Type: Integer Variable

Value: The column position within a document, where the last mouse button click occurred.

Example:

```
column mousecolumn
```

Explanation: The *column* command is used to set the current marker to the column where the mouse was last clicked.

Notes: Columns are numbered from zero.

See Also: mouseline, mousebuffer, mousewindow.

Item: mousewindow

Type: String Variable

Value: The name of the window where the last mouse button click occurred.

Example:
```
edit mousewindow
```

Explanation: The window clicked over gains the input focus and the caret, and the buffer displayed becomes the currently selected buffer.

Notes: This example would normally be used within a mouse button key definition.

See Also: mouseline, mousecolumn, mousebuffer.

Item: page

Type: Integer Variable

Value: The current page number.

Example:

```
str function "FOOTER" = $page
```

Explanation: A page footer is defined to insert the current page number into the footer line. The $ operator converts the integer *page* into a string.

Notes: If used within a dynamic text item, as above, page has the value of the page where it is inserted, other wise it has the currently selected marker's page number.

Pages are numbered from one.

See Also: line, column, pageline.

Item: pagelength

Type: Integer Variable.

Value: The number of lines on a page, as defined for the current buffer.

Example:

```
line = line + pagelength
```

Explanation: The currently selected marker is moved to the same line on the following page.

Notes:

See Also: page, line, column.

Item: pageline

Type: Integer Variable

Value: The current line number on the current page.

Example:

```
str function "pline"
  = This is line "+$pageline
```

Explanation:

A string function is defined which, when inserted as a dynamic item, will always display the correct line number.

Notes: The $ operator converts the integer value into a string.

Lines are numbered from zero at the top of each page, and number through headers and footers, etc.

See Also: line, page, column, pagelength.

Item: pages[(<buffer>)]

Type: Integer function

Value: The number of pages in the specified buffer.

Example:
```
if ( page == pages ) last=true
```

Explanation: Sets integer variable *last* to *true*, if the selected marker is on the last page.

Notes: If the parameter is not given, as in the example above, the currently selected buffer is used.

This example could be used within a header or footer definition, to select alternative text for the last page, etc.

See Also: page.

Item: pagewidth

Type: Integer Variable

Value: The width, in columns, of the current buffer.

Example:

```
margins = 10, pagewidth - 10
```

Explanation: The current margins are set ten characters in from each edge of the paper.

Notes: If no buffer is currently selected, zero is returned.

See Also: pagelength.

Item: poundwidth

Type: Integer variable

Value: By default 10, but user modifiable.

Example:

```
poundwidth = 8
```

Explanation: Sets the width of string in subsequent uses of the £ operator.

Notes: The actual width used will be expanded if the value is too large. The width must include the £ sign, the decimal point, and any commas used between thousands.

See under operators, for use of £.

See Also:

Item: query(<question>)

Type: Integer Function

Value: *True* or *false*, according to the user's response.

Example:

```
if ( query("Over write file") )
     save buffer
```

Explanation: The current buffer is saved if the user confirms the action.

Notes: A dialogue box containing the given question is displayed, allowing the user to click *Yes* or *No* buttons. If *Yes* is clicked, the function returns *true*, else it returns *false*.

It is also possible to use the **Return** key to indicate *Yes*, and the **Escape** key to indicated *No*, provided Premier has the input focus, so that the key strokes are delivered to the program.

See Also: prompt, choose.

Item: record

Type: Integer Variable

Value: The number of the current data base record.

Example:

```
if ( record > 0 ) rewind
```

Explanation: If the current data base record is not the first one, then the file is rewound to the beginning.

Notes: The first record (record zero) contains the field names, not data.

If no data base file is open, *record* will have the value zero. Use *datafile* to test if a file is open.

See Also: datafile

Item: rightmargin

Type: Integer Variable

Value: The column number of the right margin, as set for the current buffer.

Example:

```
margins = 10, rightmargin - 10
```

Explanation: Issued from within a dynamic text item, this command would set the left margin to 10, and the right margin to ten characters less than the default for the buffer.

Notes: Issued from a command file or command window, this command would set the default margins, as at the top of the document.

This variable does not supply the current margins, when used within a dynamic item.

See Also: leftmargin, tabs.

Item: serialnumber

Type: String variable.

Value: The serial number of the copy of premier being used.

Example:
```
show serialnumber
```

Explanation: The serial number is displayed.

Notes: This is also available from the icon bar *Info* option.

See Also: version.

Item: size(<array>)

Type: Integer function.

Value: The size of the named array, as given when it was created.

Example:
```
if ( i<size("a") ) a(i) = value
```

Explanation: A check is made to see if *i* is within the defined range of array elements, before an assignment is made.

Notes: Reference to non-existent elements causes an error message.

See Also: set.

Item: stddate

Type: String variable

Value: A string representing the current date.

Example:

```
fieldvalue(1) = date
```

Explanation: The current date is written to field 1 of the current data base record.

Notes: The format of this data item is designed so that normal string comparisons may be made, with the correct results, and may not be changed.

The format is e.g. 1990-06-16 17:35

This includes the time, to the nearest few seconds, (assuming your computer clock is correct!).

See Also: time, date, dateformat.

Item: substring(<string>,<index>,<length>)

Type: String function

Value: A sub-string of *string*, starting from character *index*, and having a length of *length*.

Example:
```
s = substring(text,5,3)
```

Explanation: If `text = "ABCDEFGHIJ"`
then `s = "EFG"`

Notes: If *index* is zero, then *length* characters are taken from the *end* of *string*. i.e:-

If
```
s = substring(text,0,2)
```

then `s = "IJ"`

Out of range *index* or *length* return null strings without error.

See Also: extract, index, length, locate.

Item: sum(<array>)

Type: Integer function

Value: The sum of all elements of *array*.

Example:
```
total = sum("costs")
```

Explanation: The integer variable *total* is assigned the sum of all elements of array *costs*.

Notes:

See Also: sort, set.

Item: tabset

Type: String Variable

Value: A list of tab stop columns for the currently selected buffer.

Example:
```
show tabset
```

Explanation: A string such as "8,16,24,32,40" is displayed.

Notes: If there is no selected buffer, a null string is returned.

See Also: tab, tabs.

Item: time

Type: String Variable.

Value: A string representing the current time.

Example:
```
insert "^time^"
```

Explanation: The current time is inserted as a dynamic item at the selected marker position.

Notes: The time is in the form 13:15:59

See Also: data, stddate, dateformat.

Item: type

Type: Integer Variable.

Value: The current typing mode, *true* if overtype mode is set, else *false*.

Example:

```
if ( type )
    type = insert
else
    type = over
```

Explanation:

Notes: The current typing mode is toggled between *insert* and *over* type mode.

See Also: insert, true, on, false, off.

Item: valid

Type: Integer variable

Value: Either true or false, indicating whether the copy of Premier being used has been validated or not.

Example:

```
if ( not valid ) abort
```

Explanation: If this version of Premier is not validated, the application command file is aborted.

Notes: Validated copies of Premier are ones having a valid password, entered on the *!Run* command line, or via the icon bar menu *Info* option.

This command is mainly used by commercial applications.

See Also: serialnumber.

Item: version

Type: String variable

Value: The version number of the copy of Premier being used.

Example:

```
if ( version < "Version 3.10" )
{
    show "You need version 3.1"
    abort
}
```

Explanation:

Notes: The value of *version* is as shown in this example, used to ensure that the required version of Premier is being used.

See Also: valid, serialnumber.

Item: week

Type: Integer variable

Value: The week number in the year.

Example:

```
fieldvalue(3) = $week
```

Explanation: A string version of the week number is written to field 3 of the current database record.

Notes:

See Also: weekday, month.

Item: weekday

Type: String variable

Value: The name of the current weekday.

Example:
```
if (weekday=="Saturday") GoHome
```

Explanation: If today is Saturday, the macro *GoHome* is executed.

Notes: The value of *weekday* is the full length name, including capital letter.

See Also: week, month.

Item: width

Type: Integer variable

Value: The width of the current font.

Example:

```
{width width*2}
```

Explanation: Entered as a direct dynamic command, this item doubles the width of the current font.

Notes: This example uses the *width* command, as well as the integer variable *width*.

This variable returns the width of the font at the position of the dynamic item within which it is contained, the width of the font at the position of the currently selected marker, or zero.

See Also: font, colour.

Item: window

Type: String Variable.

Value: The name of the input focus window.

Example:

```
if ( window <> mousewindow )
    edit mousewindow
```

Explanation:

If the window last clicked is not the current window, then it is selected.

Notes:

See Also: buffer, marker.

Item: windowheight

Type: Integer Variable.

Value: The height of the current window, in lines.

Example:
```
line = line + windowheight.
```

Explanation: The current marker is moved down by the height of the window. This will vary as the user re-scales the window by dragging, etc

Notes: The variable may be useful for implementing a page down command.

See Also: pagelength, pagewidth.

Item: words[(<buffer>)]

Type: Integer function

Value: The number of words in the given buffer.

Example:
```
show words("Letter")
```

Explanation: The number of words in the buffer "Letter" is displayed.

Notes: The parameter is optional, and if not given, the currently selected buffer, or the buffer containing the variable as a dynamic item, is assumed.

i.e. "This document has ^words^ words."

The number of words is counted each time it is required. This may lead to slight screen refresh delays when used as a dynamic item within very long documents.

See Also: lines, characters.

Item: year

Type: String Variable.

Value: The current year in 4 figure format.

Example:

```
if ( infoyear < year ) Archive
```

Explanation: If the value of *infoyear* is less than the current year, then the macro *Archive* is executed.

Notes: This variable is obtained from the RISC OS system variable Sys$Year, is of the form "1990"

It may be used in integer form using the # operator, e.g.

```
str "nextyear" = $(#year+1)
```

See Also: data, stddate, time.

Section 5 Command Words

This section presents all the Premier commands in alphabetical order, one per page.

All commands are **case in-sensitive,** that is, they may be given in either upper or lower case, or even in mixed case, unlike other Premier items. Also, Premier will look for macro names *before* command names, so that if a macro called "PRINT" is defined, then "print" will still be recognised as a command word, and not as the macro which must be given in upper case as defined.

For each command, the exact syntax required is shown, using the following conventions.

<name>	Items in angle brackets represent items which the user must supply, and will vary between uses of the command. The brackets are *not* part of the syntax and should *not* be typed.
[...]	Items within square brackets are optional, and need only be given when required. The brackets are *not* part of the syntax, and should *not* be typed.
command	Where this, or commandlist, is shown, either a single command, or a command list may be given. A command list must be enclosed in curly brackets. {}

The examples given are intended only as examples of the use of the appropriate command, and may not represent usable statements as they stand. For example, they may imply the pre-definition of other variables or macros, etc.

Command: * (star or asterisk)

Syntax: * <command string>

Use: To send star commands to the computer operating system.

Example: * "cat"

Explanation: The currently selected directory will be catalogued in a temporary text window.

Notes: The command should be given as a string expression, i.e. within quotes, or a string variable, etc.

The effect is the same as if the command had been given at the computer * supervisor prompt. You should avoid giving commands which may corrupt the desktop or destroy the running applications.

This facility is more useful within command files.

See Also:

Command: ABORT

Syntax: abort

Use: To abort a command file, function, macro or key definition.

Example:
```
if ( buffer == "" ) abort
```

Explanation: Used within a macro or function, etc, will abort the macro if there is no selected buffer.

Notes: This command will abort the *innermost* macro, function or command file only. i.e. it will not abort a file, if encountered within a macro in the file, and it will not abort nested macros and/or functions.

Also, command files are not aborted during initial compilation.

See Also: stop

Command: APPEND

Syntax: append

Use: To append the current data base record onto the end of the current data base file.

Example: append

Explanation: The current field values are formatted to form a new record, which is written to the end of the current data file.

Notes: The current field values are those currently held in the field variables, which may be those last read from the file, and/or new values assigned to these, or the fieldvalue(n) variables.

See Also: read, write.

Command: ASSIGN

Syntax: assign <markername>, <windowname>

Use: To assign a buffer to a window, so that the contents of the buffer are displayed in the window, when open.

Example:
```
assign "newbuff", "smallwindow"
```

Explanation: The marker "newbuffer' is assigned to the given window, so that the content of it's buffer are displayed. The marker becomes the windows cursor, and will appear as the caret when the window is selected.

Notes: Both the marker and window names may be any valid string expression.

See Also: buffer, marker, select, edit, open.

Command: BEEP

Syntax: beep

Use: To sound the computer bell tone.

Example:

```
if found beep
```

Explanation: If the system variable found is true, i.e. a recent find operation was successful, a beep is sounded.

Notes: The actual sound produced, if any, depends on the current computer configuration.

See Also:

Command: BREAK

Syntax: break <lines>

Use: To force a conditional page break.

Example:
```
macro "para"  { break 4 }
```

Explanation: A macro is defined, which when embedded as a dynamic item, will force a new page to be started if there are less than 4 lines left on the current page.

Notes: This command is only useful as a dynamic item, and will have no effect used elsewhere.

See Also: gap, newpage.

Command: BUFFER

Syntax: buffer <buffername>, [<filetype>]

Use: To create a new buffer.

Example:
```
buffer "mybuff", PremCmd
```

Explanation: A new Premier command buffer called mybuff is created.

Notes: If a buffer type of zero is given, this will be replaced by the type of the file first loaded into it, while still empty.

The file type is optional, and defaults to PremDoc when not given.

The buffer name and file type may be any valid string or integer expression, respectively.

See Also: marker, window.

Command: CASE

Syntax: case <switch>

Use: To set find and search case sensitivity on or off.

Example:
```
case on
```

Explanation: Case sensitivity is set to on.

on and *off* are system constants.

Notes: Future find or search commands will be done in case sensitive, or case-insensitive modes, according to the switch setting.

See Also: find, search.

Command: CENTRE

Syntax: centre

Use: To center text between margins.

Example:
```
{centre}
```

Explanation: Entered as a direct dynamic item, all following text on the same line will be centred between the current left and right margins.

Notes: This command is usually entered at the start of a line. If there is insufficient room, no action is taken.

See Also: right, tab, justify

Command: CHOOSE

Syntax: choose <menu>, <variable>

Use: To offer the user a wimp menu, from which to choose a text item.

Example:

```
choose "Documents", selection
```

Explanation: The menu Documents is displayed, and the user's choice assigned to the string variable selection.

Notes: The variable may be an integer type, in which case the string chosen will be converted to an integer, if possible. If not, zero will be used.

If this is used from a sub-menu macro, and the user aborts the choice by moving the pointer back over the main menu, the rest of the macro, if any, will be aborted.

If not used as a sub-menu, the pointer will be constrained to the menu area, forcing a choice.

See Also: prompt, menu.

Command: CLEAR

Syntax: clear <buffername>

Use: To clear the text from the named buffer.

Example:
```
clear "mybuff"
```

Explanation: The named buffer is cleared, and any used memory freed for further use by Premier.

Notes: The buffer is not deleted by this command.

The supplied buffer name may be formed from any valid string expression.

See Also: dispose.

Command: CLOSE

Syntax: close <windowname>

Use: To close a given window.

Example:
```
close "smallwindow"
```

Explanation: The window named smallwindow is closed, that is, it is removed from the screen.

Notes: Only really useful from a command file, as the same effect is produced by clicking the window's close icon.

The buffer contents are not lost, and the window may be re-opened intact from the icon bar menu.

The window name may be any string expression.

See Also: open.

Command: COLOURS

Syntax: colours <colour list>

Use: To set the colour of various display items..

Example:
```
colours white, red, grey, grey, white
```

Explanation: The colour of various screen items are set as required. See list below.

white & *red*, etc, are system constants.

Notes: From 1 to 7 colours may optionally be given, corresponding to the following item list.

1	Paper or window background.
2	Marked text background.
3	Margin box and perforation area.
4	Control characters.
5	Dynamic item backgrounds.
6	Hidden text, (non-printing).
7	The caret colour.

Any object may be made invisible by setting it to the colour of the paper.

The paper colour only affects windows subsequently created, allowing different colour windows to be displayed.

See Also: font.

Command: COLUMN

Syntax: column <columnnumber>

Use: To position the currently selected marker to the required column.

Example:
```
column leftmargin
```

Explanation: The currently selected marker is positioned to the left margin, on the current line. (leftmargin is a system variable).

Notes: If the required column does not exist, the marker is placed at the end of the line. The new column position becomes the prefered column.

The column number may be any integer expression.

See Also: line, cursor.

Command: COMMAND

Syntax: command

Use: To display the command window.

Example:
```
key 436 = command
```

Explanation: Function key Ctrl-Shift-F4 is assigned to open the command window, or dialogue box. (See chapter - Key Numbers.)

Notes: This command is only useful as a key definition, in order permit other commands to be entered.

See Also: datawindow, filer, finder.

Command: COMPILE

Syntax: compile <string expression>

Use: To execute a command string.

Example:

```
compile "str name = \"text\"'
```

Explanation: The command:-

```
str name = "text"
```

is executed, creating a new string variable.

Notes: Note the use of \" to include quotes within the string.

If the above compile command was included within a macro, the string variable would not be created until the macro was executed. and the variable name would be the value of name at that time.

See Also:

Command: COPY

Syntax: copy [<mark1> [, <mark2> [, <mark2>]]]

Use: To copy a section of text to another location.

Example:
```
copy marker
```

Explanation: The currently marked section is copied to the current marker position.

(*marker* is a system variable)

Notes: The copy action is different according to the number of markers given, as follows.

No markers: Marked section to PasteBuffer
1 marker: Marked section to mark1
2 markers: mark1-mark2 to PasteBuffer
3 markers: mark1-mark2 to mark3

Marker names may be any string expression.

See Also: marker, mark, cut.

Command: CURSOR

Syntax: cursor <direction>

Use: To move the currently selected marker in the required direction within the text.

Example:
```
key 397 = cursor right
```

Explanation: The cursor right key is assigned the command to move the current marker one place to the right.

(*right* is a system variable)

Notes: This command causes to marker to behave like an editting cursor. Moving from a line end causes the marker to wrap around.

When moved left or right, the new column becomes the prefered column. When moved up or down, the marker is moved to the prefered column if the next line is too short. The direction may be any integer expression.

See Also: line, column.

Command: CUT

Syntax: cut [<mark1> [, <mark2> [, <mark3>]]]

Use: To cut an area of text, and place it elsewhere.

(A move command.)

Example:
```
cut "thismark", "thatmark"
```

Explanation: The area of text between *thismark* and *thatmark* is cut out and moved to the *PasteBuffer*.

Notes: The cut action is different according to the number of markers given, as follows.

No markers: Marked section to PasteBuffer
1 marker: Marked section to mark1
2 markers: mark1-mark2 to PasteBuffer
3 markers: mark1-mark2 to mark3

Marker names may be any string expression.

See Also: copy, marker, mark.

Command: DATA

Syntax: data [<filename>]

Use: To open, or select a data file.

Example:
```
data addressfile
```

Explanation: The file given by the string variable addressfile, is opened and becomes the current data file.

Notes: Selecting a new data file will close any already open. However, when re-opened, a closed file will be re-positioned at it's previous record, and any indexes or filters are retained. Any open file will be correctly closed when Premier is terminated.

The file name may be any string expression.

If a file name is not given, the currently open file will be closed.

See Also: read, write, append, index, filter.

Command: DATAWINDOW

Syntax: datawindow

Use: To open the data base dialogue box, or window.

Example:
```
key 4 = datawindow
```

Explanation: The key Ctrl-D is defined to open the dialogue window when typed. See later for key numbers.

Notes: This command may be given in the command window, or from a command file or function key, or within a macro for inclusion in a menu, etc. The window will remain open until closed using it's close icon, or when a menu entry is de-selected.

See Also: command, filer, finder.

Command: DEFAULTS

Syntax: defaults <paper size> [, <margins>]

Use: To set the default paper size for all new documents.

Example:
```
defaults 2, 48, 50, 50, 75, 5, 70
```

Explanation: A new default paper size of 50 lines by 75 columns is defined, with 2 lines for headers and footers, no perforation skip area, and margins at columns 5 and 70.

Notes: This command will optionally accept from 1 to 7 integer expressions as follows:-

As for *paper* command:-

1	head
2	foot
3	skip
4	length
5	width

As for *margins* command:-

6	left margin
7	right margin

This command is useful in the *config* file.

See Also: paper, margins.

Command: DELETE

Syntax: delete

Use: To delete the character to the left of the currently selected marker position.

Example: key 127 = delete

Explanation: The delete key is defined to delete a character when typed. See later for key numbers.

Notes: The system variable deletedchar is assigned the character deleted.

This command may also prove useful within command macros to perform editing functions.

See Also: cut, copy.

Command: DIRECTION

Syntax: direction <searchdirection>

Use: Th set the direction for subsequent find and search operations.

Example:
```
direction forward
```

Explanation: The current find and search direction is set to forwards.

(*forward* is a system string variable)

Notes: The direction may be given as any string expression which evaluates to "forward" or "backward". System variable forward and backward have been provided to have these values.

See Also: find, search.

Command: DISPOSE

Syntax: dispose <objectname>

Use: To dispose of the named object.

Example:
```
dispose "mybuff"
```

Explanation: The buffer mybuff, and it's associated marker, is destroyed.

Notes: If used from the command window, a confirmation prompt will be given.

See Also: assign, clear.

Command: DO

Syntax: do <buffername>

Use: To cause the commands within the buffer to be executed.

Example:
```
do mycommands
```

Explanation: Commands in the buffer given by the variable mycommands are executed.

Notes: This only works if the buffer named is of type *PremCmd*.

No object file is produced.

See Also: execute.

Command: DUMP

Syntax: dump [<type>]

Use: To list Premier objects to the Journal.

Example:
```
dump "f"
```

Explanation: A list of all defined functions is written to the Journal buffer.

Notes: <type> if given, should be a single character string, denoting the type of object to be selected, as follows:-

`"i"`	Integer variables
`"s"`	String variables
`"f"`	Functions (all types)
`"m"`	Macros

If no type is specified, all objects will be listed.

See Also:

Command: EDIT

Syntax: edit <windowname>

Use: To give the named window the input focus, so that it may be edited interactively.

Example:
```
key 256 = edit mousewindow
```

Explanation: The mouse select button is defined to give the input focus to the window last clicked over.

(*mousewindow* is a system string variable)

Notes: The window name may be any string expression.

The window's cursor, i.e. the marker belonging to the buffer assigned to it, becomes the currently selected marker.

See Also: select, assign.

Command: END

Syntax: end

Use: To move the currently selected marker to the end of it's buffer.

Example:

```
end
```

Explanation: The marker is moved to the end of the buffer.

Notes: The marker is positioned beyond the last character, so that the next character inserted is added to the end of the buffer.

See Also: start.

Command: ENTER

Syntax: enter

Use: To cause a dynamic entry dialogue box to be displayed at the current marker position, or to give such a box the input focus.

Example:

```
key 386 = enter
```

Explanation: The function key F2 is defined to display the dialogue box.

Notes: If a dynamic entry dialogue box is already displayed, this command will position the caret within it.

Such a dynamic box is opened automatically if the caret is moved right onto a dynamic item. The enter command will then give this box the input focus ready for editing the dynamic item name. Circumflex characters are added to the item automatically, and should NOT be supplied.

See Also:

Command: EXECUTE

Syntax: execute <filename>

Use: To execute a named command file.

Example:
```
execute "$.MyCommands"
```

Explanation: The file MyCommands, in the root directory, is loaded and executed.

Notes: The named file's date will be compared with the compiled object code file <Premier$Dir>.Object.MyCommands, and if this is later, it will be used. Otherwise the named file will be loaded and executed. If all is well, a new object file will be created.

The file name may be any string expression.

See Also: do.

Command: EXIT

Syntax: exit

Use: To save all modified buffers, and terminate the program run.

Example:

```
exit
```

Explanation: All modified buffers are saved, and Premier removes itself from the icon bar.

Notes: If the full pathname of a file is known, this will be used, otherwise a modified buffer will be saved in the current default directory. The Journal buffer is also saved.

This command is most useful within command files which perform some function without entering interactive desktop mode.

See Also: quit.

Command: FILENAME

Syntax: filename <file name>

Use: To set the default filename for the current buffer.

Example:
```
filename "<Premier$Dir>.Data.New"
```

Explanation: The currently selected buffer has it's file name changed to that shown.

Notes: Filenames may be edited from the *Saver* dialogue box, but this allows permanent changes to be made from a command file or macro.

See Also:

Command: FILER

Syntax: filer

Use: To display the file save dialogue box.

Example:

```
macro "SaveFile" { filer }
```

Explanation: A macro is defined for possible use in a screen menu, to provide interactive file saving facilities.

Notes: This facility may be invoked from the command window, a function key, menu entry, or command file. It will remain displayed until closed using it's close icon, except from a menu, in which case it will be closed by the WIMP when the menu is re-entered.

See Also: finder, command, datawindow.

Command: FILTER

Syntax: filter <field> [,<logic>, <operator>, <data>]

Use: To set the conditions on a data base file, to define which records will be read.

Example:

```
filter 4 and equal "wilts"
```

Explanation: A filter is set on field 4 of the current data base file, so that only records containing the string wilts will be read.

Notes: The <logic> item may be and or or, and defines how this filter is used with others. For a record to be read, it must pass through all filters labelled and, and through any filter labelled or.

If only the <field> is given, all filters on that field will be removed.

If the field number is given as zero, all filters will be removed from all fields.

See Also: read, seek, index.

Command: FIND

Syntax: find <string>

Use: To search the current buffer for the given string.

Example:
```
find "Hello world"
```

Explanation: The currently selected marker is moved through it's buffer until the given string is found.

Notes: The search will be done in the currently selected direction.

Matching will be according to the current case sensitivity setting.

The string found is assigned to the system variable foundstring, and the system variable found is set true or false as appropriate.

See Also: search, case, direction, finder.

Command: FINDER

Syntax: finder

Use: To display the find and replace dialogue box.

Example:
```
key 388 = finder
```

Explanation: Function key F4 is defined to display the dialogue box when typed.

Notes: The dialogue box will remain displayed until explicitly closed, unless displayed from a menu entry, in which case it will be closed by the WIMP when the menu is re-entered, or removed.

See Also: filer, command, datawindow. find, search.

Command: FLAG

Syntax: flag <menu>, <option>, <tick>, <fade>

Use: To set option attributes in an existing menu.

Example:
```
flag "MyMenu", 5, true, false
```

Explanation: Option 5 in the menu MyMenu is marked as ticked, and unfaded.

Notes: The items <tick> and <fade> may be any logical expression. true and false are system variables.

See Also:

Command: FONT

Syntax: font <style>

Use: To select the on screen font style.

Example:
```
macro "FB" { font bold }
```

Explanation: A macro is defined which when embedded as a dynamic text item, causes all following text to be displayed in bold typeface.

Notes: This command only has any effect when given within a dynamic item.

Font styles may be added together and include colours, i.e. the following is valid:-

```
font bold + red + italics
```

Adding two colours will produce unpredictable results!

See Also: colours, send.

Command: FOOTER

Syntax: footer <footername>

Use: To set the dynamic name to be used for all page footer insertions within a document.

Example:
```
footer  "DocFoot"
```

Explanation: Any string or string function having the name DocFoot will be used for all page footers within the current document.

Notes: Any string or string function so named will be used as a footer on the bottom line of the lower page margin. Strings named as DocFoot1, 2, 3, etc, will be used on successive lines counting upwards.

This command overrides the default name of FOOTER, which is used if this command is not given.

See Also: header.

Command: GAP

Syntax: gap <lines>

Use: To cause a defined number of lines to be left clear, at a specific point in the text.

Example:
```
macro "TableSpace" { gap 10 }
```

Explanation: A macro is defined which, when used as a dynamic item, will cause at least a 10 line gap to be left following the point of insertion.

Notes: If there are not enough lines left on the current page, a gap of the defined size will be left at the top of the next page.

This command only has any effect when issued from within a dynamic item.

See Also: break.

Command: GOTO

Syntax: goto <markername>

Use: To cause the named marker to become the currently selected marker, and the current window cursor, or caret.

Example:
```
goto "SavedPosition"
```

Explanation: The marker SavedPosition becomes the window caret.

Notes: When a buffer is assigned to a window, the buffer's default marker, having the same name as the buffer, will be used as the caret when the window is opened. This command allows any other named markers within the buffer to be used instead.

The displayed caret may be moved between several named markers using this command.

See Also: marker, select.

Command: HEADER

Syntax: header <headername>

Use: To set the dynamic name to be used for all page header insertions within a document.

Example:
```
header "DocHead"
```

Explanation: Any string or string function having the name DocHead will be used for all page headers within the current document.

Notes: Any string or string function so named will be used as a header on the top line of the upper page margin. Strings named as DocHead1, 2, 3, etc, will be used on successive lines counting downwards.

This command overrides the default name of HEADER, which is used if this command is not given.

See Also: footer.

Command: HIDE

Syntax: hide <state>

Use: To prevent selected text from being printed.

Example:

```
str "comment" = "^{hide on}
                ^Silly!^{hide off}^"
```

Explanation: Entered as a dynamic text item, this string will be seen on the screen, but not sent to the printer.

Notes: Non-printing comments like this are best placed on blank lines, to prevent paragraph formatting being adversely affected.

Nothing between hide on and hide off will be printed, including newlines.

See Also:

Command: IF

Syntax: if (<condition>) <command1>
[else <command2>]

Use: To cause a list of commands to be conditionally executed, or not.

Example:
```
if ( fontbold ) send boldon
                else send boldoff
```

Explanation: If bold font is currently selected, the string defined by boldon is sent to the printer, otherwise the string defined by boldoff is sent.

Notes: <command1> or <command2 > may be command lists enclosed within curly brackets {}

The *else* part is optional.

The condition may be any integer expression, and the following command, or command list, will be executed if this expression is non-zero.

See Also: repeat, while.

Command: INDEX

Syntax: index <fieldnumber>

Use: To create an index for the current data base file, based on the given field number.

Example:
```
index 4
```

Explanation: An index is created, listing all records in alphabetical order on the content of field 4.

Notes: Only records passing through any currently set filters will be included in the index.

All the data from the selected field is stored in memory with pointers to the disc records, permitting fast access during seek commands.

All future read operations will be done in the indexed order.

See Also: data, filter, read, seek.

Command: INT

Syntax: int <name> = <value>

Use: To create a new integer variable, and initialise it with a given value.

Example:
```
int "fred" = 37
```

Explanation: A new integer variable called fred is created, and assigned the value 37.

Notes: If fred already exists, no error is generated, and the existing variable is assigned the new value.

The variable name may be any string expression, allowing names to be made up;-

```
int "fred" + $numb = 0
```

See Also: function, str.

Command: INT ARRAY

Syntax: int array <name>(<size>)

Use: To create an integer array.

Example:
```
int array "new"(20)
```

Explanation: A new integer array called new is created having 20 elements.

Notes: All elements are initialised to zero.

See Also: set, sort.

Command: INT FUNCTION

Syntax: int function <name> [<command-list>]
= <integer-expression>

Use: To define an integer function.

Example:
```
int function "Vat" = IP(1)*vat/100
```

Explanation: An integer function called "Vat" is defined, which gives the V.A.T. due on an amount supplied as a parameter. The variable vat is assumed to define the percentage rate used.

Notes: See the Premier User's Guide for details concerning the use of passed parameters.

The optional command list, if required, must be enclosed in curly brackets.

See Also:

Command: INSERT

Syntax: insert <string> [,<marker>]

Use: To insert a text string into a buffer.

Example:
```
insert date, lastmark
```

Explanation: The current date is inserted, as a fixed text string, at the position of the marker lastmark.

Notes: The string may be any string expression.

If the marker name is not provided, the currently selected marker will be used.

This command is used in key definitions to cause the qwerty keys to have the desired actions.

See enter, for dynamic date insertion.

See Also: enter, delete.

Command: JUSTIFY

Syntax: justify <state>

Use: To turn text justification on or off.

Example:
```
macro "JustOn" { justify on }
```

Explanation: A macro is defined which, when embedded as a dynamic text item, will cause all following text to be fully justified between margins, until explicitly turned off by another suitable macro.

Notes: This command only has effect when issued from within a dynamic item.

The <state> may be any integer expression, non-zero being equivalent to on, zero to off. The system variables on and off may be used for convenience.

See Also: font, gap, break, margins.

Command: KEY

Syntax: key <number> = command

Use: To define a key to execute the given command(s) when typed.

Example:
```
key 65 = insert "A"
```

Explanation: The keyboard key labelled A is defined to insert an A at the currently selected marker.

Notes: All keys must be defined before they may be used, usually in a command file executed at start up.

Any key, with shift, control, or both, may be defined, in addition to the mouse buttons.

The command may be a command list or macro.

A full list of key numbers is given in a later chapter

See Also:

Command: LINE

Syntax: line <linenumber>

Use: To move the selected marker to the given line.

Example:
```
line 56
```

Explanation: The current marker is moved to line 56.

Notes: Lines are numbered from zero, and numbers run consecutively through header and footer margins, and perforation skip areas. It is not possible to place the marker in one of these areas, and the nearest will be chosen.

This command is most useful within editing command files.

See Also: column, cursor.

Command: LINK

Syntax: link <field>, <file>, <keyfield>

Use: To link two data files.

Example:
```
link 1,"$.Accounts",4
```

Explanation: Field 1 in the current data file is linked to field 4 in file $.Accounts.

Notes: Any number of links may be formed on any field.

When a link has been defined, every access to the first file, will cause the second file to be searched for a field which matches the link field in the first file.

There must be a *current* data file, and the second file must have been *seen*, i.e. previously opened, before linking. e.g.

```
data "$.SecondFile"
data "$.FirstFile"
link  2, "$.SecondFile", 6
```

Links are not saved between sessions.

See Also: filter, read, index.

Command: LOAD

Syntax: load <filename>

Use: To load the given file from disc into the currently selected buffer.

Example:
```
load "NewFile"
```

Explanation: The file NewFile is inserted at the currently selected marker.

Notes: If the buffer was empty, the marker will be left at the start, otherwise it will be left after the inserted text, as with any other text insertion operation.

Only *Premier*, or *text*, type files may be loaded.

See Also: insert, save.

Command: MACRO

Syntax: macro <name> { <commandlist> }

Use: To define a named macro.

Example:
```
macro "LQ" { send LetterQual }
```

Explanation: A macro called LQ is defined to send the string variable LetterQual to the printer. This would be used as a dynamic item to turn letter quality printing mode on, assuming the variable is defined to have the correct codes.

Notes: A macro definition always expects a command list, and the curly brackets are therefore always required.

A macro given the same name as an existing command will replace that command. This should be avoided. Note however that command words may be given in either case, while macros are case sensitive. e.g. A macro called Save will not replace the command, provided the command is given in lower case.

See Also: str function.

Command: MARGINS

Syntax: margins <left> , <right>

Use: To define the left and right page margins.

Example:

```
macro "indent" {margins 10,50}
```

Explanation: A macro is defined which, when embedded as a dynamic item, will cause the following lines to have the left margin to be 10 characters wide, and the printed text width to be 40 characters wide.

Notes: When given as a dynamic item as above, this command only affects the following lines. If given from the command window, or within a command file, etc, this command defines the default margins to be used from the top of the document.

Column numbers start from zero, so that text starts on the given left margin number.

See Also: paper, defaults.

Command: MARK

Syntax: mark

Use: To place an un-named marker, to define one end of a marked section of text.

Example:
```
key 390 = mark
```

Explanation: Function key F6 is defined to place a marker.

Notes: This command defines one end of a marked (coloured) screen area, which may be used with the cut and copy commands.

If two marks are already placed, this command moves the nearest mark to the current caret position.

See Also: marker, cut, copy.

Command: MARKER

Syntax: marker <name>

Use: To place a named marker at the position of the currently selected marker.

Example:
```
marker "here"
```

Explanation: A marker called here is placed at the current position.

Notes: Named markers may be explicitly used in the cut and copy commands, or used as a window cursor using the goto command.

If the named marker already exists, it will be moved to the new position without error.

The marker name may be any string expression.

See Also: mark, cut, copy, paste, goto.

Command: MENU

Syntax: menu <name> = <itemlist>

Use: To define a WIMP type screen menu.

Example:
```
menu "Actions" = "!Move,Scrap"
```

Explanation: A menu is created having the title Actions, and the menu options Move and Scrap.

The Move option is shown ticked.

Notes: The menu options may be any string, but an error will result if a suitably named object cannot be found when the option is selected. The following prefixes may be used.

! Tick option.
~ Fade option (non-selectable)
> Allow dialogue box selection mechanism.

In addition, the '|' (bar) may be used in place of a comma, to separate with a dotted line.

See Also: offer, choose, flag.

Command: MESSAGES

Syntax: messages <level>

Use: To set the severity level below which messages are not to be reported.

Example:
```
messages verbose
```

Explanation: All future messages of this level or higher will be reported.

Notes: Message levels may be set using the followinf system variables.

verbose	Reporting of almost all actions.
info	Information messages.
warning	Some expected action did not happen.
error	Usually a user command error.
serious	Future actions may not work.
fatal	The program must terminate.
bug	A program bug has occurred, but Premier will try to continue.

See Also:

Command: OFFER

Syntax: offer <menu>

Use: To display a defined menu for the user's choice of action.

Example:
```
key 257 = offer "MyMenu"
```

Explanation: The mouse menu button is defined to offer the menu called MyMenu.

Notes: The menu will be removed when an option is selected. An error is given if the named selection is neither a menu or a macro.

See Also: choose, menu.

Command: OPEN

Syntax: open <window>

Use: To open a defined window.

Example:
```
open "NewWindow"
```

Explanation: The named window is displayed on screen.

Notes: The named window will be displayed on the screen, but not given the input focus, which may be done using the edit command.

The window name may be any string expression which evaluates to a defined window.

See Also: window, close, edit.

Command: PAGE

Syntax: page 7

Use: To number a page.

Example: {page 10}

Explanation: Embedded as a dynamic item, this command will cause the current page to be number 10, overriding the original number.

Notes: Following pages will continue to be numbered in sequence, from the new number given.

Any dynamic page numbers *before* the embedded command will continue to show the *original* page number, while those *following* will show the *new* number.

See Also:

Command: PAPER

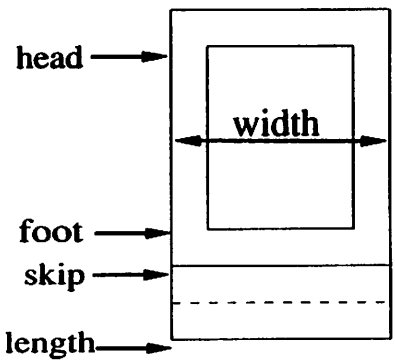
Syntax: paper <head>, <foot>, <skip>, <length> [,<width>]

Use: To define line numbers at which the page top and bottom margins, and perforation skip areas start.

Example:
```
paper  2, 62, 64, 70, 80
```

Explanation: A paper size of 70 lines x 80 characters is defined, with the text area of 60 characters, and header and footer margins of 2 lines. N.B. lines number from zero, so that length points to the top of the next page.

Notes:



The new paper size is applied to the document containing the currently selected marker.

See Also: margins, defaults.

Command: PASTE

Syntax: paste [<marker>]

Use: To copy the text contained in the PastBuffer, to a document buffer.

Example:
```
paste "LastMark"
```

Explanation: The contents of the PastBuffer are copied to the position of the marker LastMark.

Notes: If no marker is given, the currently selected marker is used.

See Also: cut, copy, marker.

Command: PAUSE

Syntax: pause <state>

Use: To cause the printer to wait after each page is printed.

Example:
```
pause on
```

Explanation: Pause mode is switched on.

Notes: The default mode is pause off.

on and *off* are system variables.

See Also: print

Command: PRINT

Syntax: print [<copies> [,<from> [,<to>]]]

Use: To send the contents of the currently selected buffer to the printer.

Example:
```
print  1, 5, 5
```

Explanation: One copy of page 5 from the currently selected buffer is printed.

Notes: If a string variable PIN is defined, this will be sent to the printer before any text, while any send commands within dynamic items will also be executed. If string variable POUND is defined, the contents of this will be sent in place of any £ characters.

See Also: printmarked, pause.

Command: PRINTMARKED

Syntax: printmarked

Use: To print the currently marked text, if any.

Example:
```
key 400 = printmarked
```

Explanation: The Shift-Print key is defined to print any marked section of text.

Notes: A newline is appended to the marked section, to ensure the output is printed.

See Also: print, printout.

Command: PRINTOUT

Syntax: printout <string expression>

Use: To send a text string directly to the printer.

Example:
```
printout fieldlength(2)<fieldvalue(2)
```

Explanation: Field 2 from the currently selected data file is printed, left justified in it's own defined fieldwidth. i.e. Padded with spaces to the right.

Notes: No newline character is added, so that further output may be printed on the same line. Newlines should be sent by the user when required.

Any valid string expression may be used, including any font control variables etc.

See Also: print, printmarked.

Command: PROMPT

Syntax: prompt <message>, <variable>

Use: To prompt the user with a message, and assign a variable with the value of any response.

Example:
```
prompt "File name ?", filename
```

Explanation: The user is prompted to enter a file name, and this is assigned to the string variable filename.

Notes: The variable may be string or integer. If an integer variable is given, any leading digits in the typed response will be used, or zero assumed.

The current value of the variable will be displayed in the prompt window, and may be used or edited by the user.

See Also: choose, show.

Command: QUIT

Syntax: quit

Use: To remove Premier from the icon bar.

Example:
```
quit
```

Explanation: Premier is terminated.

Notes: This command is intended to be used within command files, where file saving etc. is defined. No warning is given if modified files exist.

Use the icon bar menu for normal quitting.

See Also: exit.

Command: READ

Syntax: read

Use: To read the next record from a data base file.

Example:

```
read
```

Explanation: The next record is read from the currently selected data base file.

Notes: The record will be the next in any index, or the next in the file, after applying any filter tests. The fields within the new record will be assigned to the field variables, as defined in the first record.

The system variable found will be assigned true or false as appropriate.

See Also: write, seek, index, filter, rewind.

Command: RECORD

Syntax: record

Use: To start recording key strokes for later replay.

Example:

```
key 18 = record
```

Explanation: Key **Ctrl R** is defined to start the recording process.

Notes: Recording will continue until the command is repeated, or the **replay** command is given. Mouse button keys are recorded.

This facility is useful for repeating an edit.

See Also: replay.

Command: REPEAT

Syntax: repeat <command> until <condition>

Use: To repeat a command, or command list, until some specified condition is met.

Example:
```
repeat { read print } until not found
```

Explanation: The current data file is read, and the current document printed, until there are no more records.

Notes: The document is presumed to contain data base fields as dynamic text items, perhaps for a mail merge operation.

The command may be a single command, or a command list within curly brackets as shown.

See Also: while.

Command: REPLACE

Syntax: replace <string>

Use: To replace text located with the find or search commands with the string given.

Example:
```
replace "New text"
```

Explanation: The most recently found string is replaced by the new text.

Notes: The result of a find or search command causes the string variable foundstring to be assigned the text found. This command expects this string to be at the current marker position, and will not carry out the replacement if this is not the case.

See Also: find, search.

Command: REPLAY

Syntax: replay

Use: To replay a recorded key stroke sequence.

Example:
```
replay
```

Explanation: The last recorded key sequence is replayed, as though the keys were being re-typed.

Notes: This command also terminates the record process, if active, to avoid recursion.

See Also: record.

Command: RESIZE

Syntax: resize <window>, <top>, <bot>, <left>, <right>

Use: To resize an existing window.

Example:
```
resize "View", 1000,500,200,800
```

Explanation: The window called View is positioned on the screen at the co-ordinates given.

Notes: If the window is not currently open, it is opened.

The window name may be any string expression, and the co-ordintates any integer expression.

Co-ordinates are screen OS units -
0 - 1279 in x
0 - 1023 in y
with the origin at the bottom left of the screen.

See Also: Window.

Command: REWIND

Syntax: rewind

Use: To rewind the current data file to the start.

Example:
```
rewind
```

Explanation: The current data base file is returned to the first record, and a read operation performed.

Notes: It an index exists, the first record in this will be re-selected, otherwise the first actual record, containing the field names will be read.

See Also: read, index, filter.

Command: RIGHT

Syntax: right

Use: To right justify text to the right margin.

Example: `{right}`

Explanation: Entered as a direct dynamic command, all following text on the same line will be positioned up to the right margin.

Notes: If the current column position is too great, no action is taken.

The command has no meaning if not given from within a dynamic item.

See Also:

Command: SAVE

Syntax: save <buffer>, <file>

Use: To save a named buffer as a named file.

Example:
```
save buffer, "BackUpFile"
```

Explanation: The currently selected buffer is saved in a file called BackUpFile.

Since a full pathname is not given, the currently selected directory will be used.

Notes: (*buffer* is a system variable)

Both the buffer and file names may be any string expression.

See Also: load.

Command: SCROLL

Syntax: scroll <lines> <columns>

Use: To set the window scroll margins for the window displaying the currently selected buffer.

Example:

```
scroll 3, 5
```

Explanation: The scroll margins are set to 3 lines, 5 columns.

Notes: The scroll margins specify the number of lines and columns from the edge of the window at which scrolling starts when the caret is moved.

See Also:

Command: SEARCH

Syntax: search <string>

Use: To search for a given text string, using special pattern matching characters, as listed below.

Example:
```
search "^&"
```

Explanation: A search is made, in the current direction, for a string of one or more alphabetic characters.

The full range of wild card characters follows-

Notes:

%	any printable character
&	any alphabetic character
#	any digit
!	any punctuation
. (dot)	any character
@	any alphanumeric or underscore
$	a newline character
(space)	any space, tab, newline, etc.
+	any upper case letter
-	any lower case letter
|A	the contol character Ctrl-A, etc.
\c	matches literal 'c', if c is special.
*c	zero or more repetitions of 'c'.
^c	one or more repetitions of 'c'.
~c	any character which is not 'c'.
[abc]	any character from set a,b,c, etc.
"	toggles case sensitivity.

See Also: find, case, direction, finder.

Command: SEEK

Syntax: seek <field> <string>

Use: To search for a data base record having the given string in the specified field.

Example:
```
seek 3 "jones"
```

Explanation: A record in which field 3 contains jones will be located.

Notes: Searching is via the index if one exists. Note that if the index has been created on another field, then searching will have to take place on the disc, which will be searched in indexed order, and therefore take longer than if an index had not been created!

Only records passing through any set filters will be examined.

See Also: index, filter, read.

Command: SELECT

Syntax: select <marker>

Use: To select a marker for many subsequent editing operations.

Example:
```
select "MyBuffer"
```

Explanation: The marker called MyBuffer, presumably this buffer's default marker, is selected for future operations.

Notes: The named marker becomes the currently selected marker, as referred to in other commands, such as insert, delete, etc.

See Also: buffer, marker.

Command: SEND

Syntax: send <string>

Use: To send the given string of characters to the printer.

Example:
```
send "\27R"
```

Explanation: The sequence of ASCII characters 27, 82, is sent to the printer. That is, the escape character, and the letter 'R'.

Notes: See the Premier User's Guide for full details of embedding control characters in strings.

This command will only function when the printer is active, and is normally used from within a dynamic item.

See Also: print.

Command: SET

Syntax: .set <array>, <value>

Use: To set all elements of an array to some value.

Example:
```
set "ar", 0
```

Explanation: All elements of the integer array *ar* are set to zero.

Notes: The array may be integer or string, and the value will be interpreted accordingly.

<array> may be any string expression.

See Also:

Command: SETTYPE

Syntax: settype <filetype>

Use: To set the file type of the currently selected buffer.

Example:
```
settype premdoc
```

Explanation: The current buffer is given the file type of *PremDoc*, or hex D26.

(*premdoc* is a system variable)

Notes: The file type may be given as any integer expression.

See Also:

Command: SHOW

Syntax: show <item> [, <itemlist> [, ...]]

Use: To display the values of various items, or give messages from within command files.

Example:
```
show "The time is now", time
```

Explanation: The message "The time is now 17:32", for example, is displayed in a message window, and to the Journal.

Notes: Multiple items must be separated by commas, and spaces are automatically inserted between the items displayed.

Items may be mixed integer or string, literals, or variables, etc.

See Also: prompt.

Command: SKIP

Syntax: skip <string>

Use: To cause the marker to move over the given string.

Example:
```
skip foundstring
```

Explanation: The string found during the last find or search operation is skipped over.

(*foundstring* is a system string variable)

Notes: The given string must exist at the current marker position, or no action is taken.

See Also: find, search, replace.

Command: SORT

Syntax: sort <array>

Use: To sort an array.

Example:
```
sort letters
```

Explanation: An array named *letters* is sorted into order.

Notes: The array may be integer or string. A string array will be sorted according to the ASCII code sequence.

See Also: set.

Command: SPOOL

Syntax: spool <filename>

Use: To save the current buffer in the named file, as plain text, formatted as it would be printed.

Example:
```
spool "TextFile"
```

Explanation: The current buffer is saved as a text image.

Notes: If the file is of type *PremDoc*, all dynamic items will be translated correctly.

See Also: save, print.

Command: START

Syntax: start

Use: To set the current marker to the start of it's buffer.

Example:
```
start
```

Explanation: The currently selected marker is set to the start of it's buffer.

Notes:

See Also: end.

Command: STOP

Syntax: stop

Use: To terminate a *repeat* or *while* loop.

Example:

```
repeat
{    ...
     if (not found) stop
     ...
}
```

Explanation:

Notes: Execution will continue with the statement immediately following the loop, if any.

If loops are nested, only the inner one is stopped.

See Also: abort, while, repeat.

Command: STR

Syntax: str <name> = <string>

Use: To create a new string variable, and initialise it's value.

Example:

```
str "bill" = "I'm string Bill"
```

Explanation: A new string variable called bill is created, and given the initial value as shown.

Notes: It a string variable called bill already exists, no warning will be given, but the new value assigned to the existing variable.

The variable name, and the new value, may be any string expression. This permits the name to be name to be manufactured, i.e.-

```
str "String" + $i = ""
```

See Also: str function, str array.

Command: STR ARRAY

Syntax: str array <name>(<size>)

Use: To create a string array.

Example:

```
str array "Files"(10)
```

Explanation: A new string array called files is created, having 10 elements.

Notes: All elements are initially set to be null strings.

Array elements may be set using:-

```
Files(3) = "MyFile"
```

See Also: set, size, sort.

Command: STR FUNCTION

Syntax: str function <name> [<comands>] = <value>

Use: To create a new string function.

Example:

```
str function "jim" = £(IP(1)*vat)
```

Explanation: A new string function called jim is created.

This multiplies the first passed parameter by the integer variable vat, and formats the result as a money string.

Notes: If a function called jim already exists, no warning is given, and the new definition used to replace the old one.

The item <commands> is optional, but should be a command list between curly brackets, if given.

See Also: str, int, str array..

Command: SWAPCASE

Syntax: swapcase

Use: To toggle the case of the character at the current marker, between upper and lower case.

Example:

```
key 3 = swapcase
```

Explanation: The key Ctrl-C is defined to change the case of the current character.

Notes: If the current character is not alphabetic, no action is taken and no error is issued.

See Also:

Command: SYNTAX

Syntax: syntax <command>

Use: To display the exact syntax of a command.

Example:
```
syntax "save"
```

Explanation: The correct syntax of the save command is displayed.

Notes: Note that the command must be in the form of a string expression, so that the above example requires quotes.

The items within the displayed syntax may be used individually with another syntax command, to break down the items to their simplest form.

See Also:

Command: TAB

Syntax: tab <column>

Use: To position following text at a specific column position.

Example: `{tab25}`

Explanation: Used as a direct dynamic command, this item will cause the next character to be printed in column 25.

Notes: This command has no effect unless given from within a dynamic text item.

This command should not be confused with TABS, which sets the current tab stops.

See Also: centre, right.

Command: TABS

Syntax: tabs <tab stop list>

Use: To define tab stop columns.

Example:
```
tabs 10,20,30
```

Explanation: The first three tab stops are set to the columns shown. The rest are set to zero, and will be ignored.

Notes: If this command is given from a command file, or the command window, etc, the default tabs for the current buffer are set.

If the command is included within a dynamic item, the new tab stops apply from the current position onwards.

Up to 16 tab stops may be defined.

The tab list may be any valid string expression evaluating to a sequence of comma separated integers.

See Also:

Command: TYPE

Syntax: type <mode>

Use: To set the typing mode to insert, or overtype.

Example:
```
type over
```

Explanation: The current typing mode is set to overtype, so that new characters replace the existing ones, rather than being inserted between them.

Notes: (*over* is a system variable)

The mode may be any integer expression, but the system variables over and insert have been defined for convenience.

See Also:

Command: UNDERSPACES

Syntax: underspaces <mode>

Use: To select whether spaces should be underlined, within underlined font.

Example:
```
underspaces true
```

Explanation: After giving this command, all underlined font will be displad with spaces underlined.

Notes: This command has no effect on printed output, but is provided so that the screen display may match the printer underline style available.

See Also: font.

Command: UNMARK

Syntax: unmark

Use: To unmark a text area created with the mark command.

Example:

```
key 406 = unmark
```

Explanation: The key Shift-F6 is defined to clear any marked screen area.

Notes: Both the markers are removed.

See Also:

Command: UPDATE

Syntax: update [<marker1>, <marker2>]

Use: To refresh all windows displaying the currently selected marker's buffer.

Example:

```
update "Mark1", "Mark2"
```

Explanation: The text area between the two named markers is redrawn.

Notes: This action may be necessary where a new data base record has been read, and the current window displays data base fields as dynamic items.

If no markers are given, the whole of the currently selected buffer will be redrawn.

See Also:

Command: VIEW

Syntax: view <file name>

Use: To view a given file in a new window.

Example:
```
view "$.Letters.LastOne"
```

Explanation: A window containing the given file is displayed on the screen.

Notes: This command is equivalent to:-

```
str "name"=extract(<filename>,0,".")
buffer name
window name
assign name, name
select name
load name, <filename>
open name
```

It is exactly equivalent to dragging the named file onto the Premier icon on the icon bar.

See Also: buffer, window, assign, load, edit.

Command: WAIT

Syntax: wait <time>

Use: To cause Premier to enter a wait state for the given length of time.

Example:

```
wait 20
```

Explanation: Program execution is suspended for 20 seconds.

Notes: Other desk top applications will continue.

This command is only useful within command files, for example, to construct an automated display routine, etc. where normal action would be too fast.

See Also:

Command: WIDTH

Syntax: width <size>

Use: To specify the width of all following text characters.

Example: `{width32}`

Explanation: Embedded as a direct dynamic command, this item will cause all following text to be displayed at twice the normal width.

Notes: Normal characters are 16 wide.

Although any size may be used, sized chosen should match your printer's expanded and condensed type faces.

This command has no effect if not given from with a dynamic item.

See Also:

Command: WHILE

Syntax: while (<condition>) <command list>

Use: To cause a command list to be repeated, so long as a given condition exists.

Example:

```
while ( i < 10 ) { ar(i) = i }
```

Explanation: All elements of array ar, from i to 10, are set to the corresponding value of i.

If i is already equal to, or greater than 10, nothing is done.

Notes: The condition may be any logical expression, and is evaluated *before* any commands are executed. If the condition is *true*, the command list is executed once, and the condition re-evaluated. If it is *false*, the loop terminates.

See Also: repeat.

Command: WINDOW

Syntax: window <name>
[<top>, <bottom>, <left>, <right>]

Use: To create a new window.

Example:
```
window "NewView"
```

Explanation: A window called NewView is created having default size and position.

Notes: The optional values specify the screen positions of the window edges in screen coordinates. These range from 0 to 1279 in the x direction, and 0 to 1023 in the y direction.
The origin (0,0) is at the lower left corner of the screen.

By default, windows are stacked at the top right of the screen, each slightly lower than the last. From here, any window may be quickly found, and expanded to full size and back.

See Also: open, assign, close, edit.

Command: WRITE

Syntax: write

Use: To write the current data base record back to disc.

Example:
```
write
```

Explanation: The current field variables are formatted into a new record, and written to where the last record was read from.

Notes: The fields are padded, or truncated in length, as required, according to the original length, defined by the field headers in record zero.

See Also: read, append.

Command:

Syntax:

Use:

Example:

Explanation:

Notes:

See Also:

Section 6 Key Codes

This section lists the numbers by which the keyboard keys are known, and which must be used when defining their actions. These values follow the ASCII code for keys marked with printable characters.

The following tables show the codes for each key when used singly, and in combination with the shift and control keys, or with both where appropriate.

The Number Keys.

Key	Code	Shifted
0	48	41
1	49	33
2	50	64
3	51	35
4	52	36
5	53	37
6	54	94
7	55	38
8	56	42
9	57	40

Punctuation Keys.

Key	Code	Shifted	Control	
space	32			
!	33			
"	34			
#	35			
$	36			
%	37			
&	38			
'	39	34		
(	40			
)	41			
*	42			
+	43			
,	44	60		
-	45	95		
.	46	62		
/	47	63		
:	58			
;	59	58		
<	60			
=	61	43		
?	63			
@	64			
[	123	91	27	(escape)
\	124	92	28	
]	125	93	29	
^	126	94	30	
_	127	95	31	

Alphabetic Keys.

Key	Code	Shifted	Control	
a	97	65	1	
b	98	66	2	
c	99	67	3	
d	100	68	4	
e	101	69	5	
f	102	70	6	
g	103	71	7	
h	104	72	8	(backspace)
i	105	73	9	(tab)
j	106	74	10	(newline)
k	107	75	11	
l	108	76	12	(newpage)
m	109	77	13	(return)
n	110	78	14	
o	111	79	15	
p	112	80	16	
q	113	81	17	
r	114	82	18	
s	115	83	19	
t	116	84	20	
u	117	85	21	
v	118	86	22	
w	119	87	23	
x	120	88	24	
y	121	89	25	
z	122	90	26	

Function Keys,

Key	Code	Shifted	Control	Both
Print	384	400	416	432
F1	385	401	417	433
F2	386	402	418	434
F3	387	403	419	435
F4	388	404	420	436
F5	389	405	421	437
F6	390	406	422	438
F7	391	407	423	439
F8	392	408	424	440
F9	393	409	425	441
F10	458	474	490	506
F11	459	475	491	507
F12	460	476	492	508

Cursor Keys

Key	Code	Shifted	Control	Both
Left	396	412	428	444
Right	397	413	429	445
Down	398	414	430	446
Up	399	415	431	447
Home	30			
Delete	127			
Insert	461	477	493	509
Copy	395	411	427	443
Pg. Down	414	430	446	462
Pg. Up	415	431	447	436
Tab	394	410	426	442

The Mouse Buttons.

Mouse buttons may be programmed as any other key, by using the numbers below. However, the following should be understood when programming double clicks, or drag codes.

When a button is double clicked, two button codes are generated, and therefore two programmed actions may take place. Firstly, any action assigned to the normal button code will be executed. Secondly, *if the button is clicked again within about half a second*, any action assigned to the double click code will be executed.

The same happens with the drag codes listed below. Firstly any action assigned to the normal button code will be executed. Then, *if the key is held down*, any action assigned to the drag code will be executed.

The menu button has no double click, or drag code.

Button	**Code**	**Double**	**Drag**
select	256	261	259
menu	257		
adjust	258	262	260

Index

! 167
7, 167
$ 7, 167
% 5, 7, 167
& 7, 167
( ) 5
* 5, 167
*= 5
+ 5-6, 167
+= 5-6
- 5, 167
-= 5
. 167
/ 5
/= 5
< 6
<= 6
<> 6
= 5-6
== 6
> 6
>= 6
? 7
?+ 7
@ 167
| 167
\ 167
] 167
^ 167
| 7, 167
~ 167
" 167
£ 7

A

ABORT 86
adjust 202
AND 6, 10
APPEND 86
ARRAY 132, 180
ASSIGN 88

B

backspace 199
BEEP 89
black 10
blue 10
bold 10
BREAK 90
buffer 13
BUFFER 91
bug 145

C

CASE 92
CENTRE 93
Centre justify 6
character 14
characters 15
CHOOSE 94
CLEAR 95
CLOSE 96
colour 16
COLOURS 97
column 17
COLUMN 98
command 10, 99
COMPILE 100
Concatenation 6
content 18
COPY 101
CURSOR 102
CUT 103

D

DATA 104
datafile 19
DATAWINDOW 105
date 20
dateformat 21
DEFAULTS 106
defined 22
DELETE 107

deletedchar 23
direction 24
DIRECTION 108
DISPOSE 109
DO 110
down 10
DUMP 111

E

EDIT 112
end 25
END 113
ENTER 114
equal 10
Equal to 6
error 10, 145
errors 26
escape 198
EXECUTE 115
EXIT 116
Extension 6
extract 27

F

false 10
fatal 10, 145
fieldcount 28
fieldlength 29
fieldname 30
fieldvalue 31
filename 32
FILENAME 117
FILER 118
filetype 33
FILTER 119
FIND 120
FINDER 121
FLAG 122
font 34
FONT 123
fontbold 35
fontitalic 36
fontsubscript 37
fontsuperscript 38
fontunderline 39
footer 40
FOOTER 124
found 41
foundstring 42
FUNCTION 133, 181

G

GAP 125
GOTO 126
Greater than 6
Greater than or equal to 6
greaterthan 10
green 10
grey 10

H

header 43
HEADER 127
HIDE 128

I

IF 129
index 44
INDEX 130
info 10, 145
insert 10, 134
INT 131, 133
inverse 10
italic 10

J

JUSTIFY 135

K

KEY 136

L

left 10
Left justify 6
leftmargin 46
length 45
Less than 6
Less than or equal to 6
lessthan 10
line 48
LINE 137
lines 47
LINK 138
LOAD 139
locate 49

M

MACRO 140
MARGINS 141
MARK 142
marked 50
marker 51
MARKER 143
MENU 144, 202
messages 52
MESSAGES 145
month 53
mousebuffer 54
mousecolumn 55
mousewindow 56

N

Negation 6
newline 10, 199
newpage 10, 199
NOT 6
Not equal to 6
notequal 10

O

obey 10
off 10
OFFER 146
on 10
OPEN 147
OR 6, 10
orange 10
over 10

P

page 57
PAGE 148
pagelength 58
pageline 59
pages 60
pagewidth 61
PAPER 149
PASTE 150
PasteBuffer 101, 103
PAUSE 151
poundwidth 62
premcmd 10
premdata 10
premdoc 10
PRINT 152
PRINTMARKED 153
PRINTOUT 154
PROMPT 155

Q

query 63
QUIT 156

R

READ 157
record 64
RECORD 158
red 10
REPEAT 159
REPLACE 160
REPLAY 161
RESIZE 162
return 199
REWIND 163
right 10

RIGTH 164
Right justify 6
rightmargin 65

S

SAVE 165
SCROLL 166
SEARCH 167
SEEK 168
SELECT 169, 202
SEND 170
serialnumber 66
serious 10, 145
SET 171
SETTYPE 172
SHOW 173
size 67
SKIP 174
SORT 175
SPOOL 176
star 85
START 177
stddate 68
STOP 178
STR 179-181
subscript 10
substring 69
sum 70
superscript 10
SWAPCASE 182
SYNTAX 183

T

TAB 184, 199
TABS 185
tabset 71
text 10
time 72
true 10
type 73
TYPE 186

U

underline 10
UNDERSPACES 187
UNMARK 188
up 10
UPDATE 189

V

valid 74
verbose 10, 145
version 75
VIEW 190

W

WAIT 191
warning 10, 145
week 76
weekday 77
WHILE 193
width 78
WIDTH 192
window 79
WINDOW 194
windowheight 80
words 81
WRITE 195

Y

year 82
yellow 10